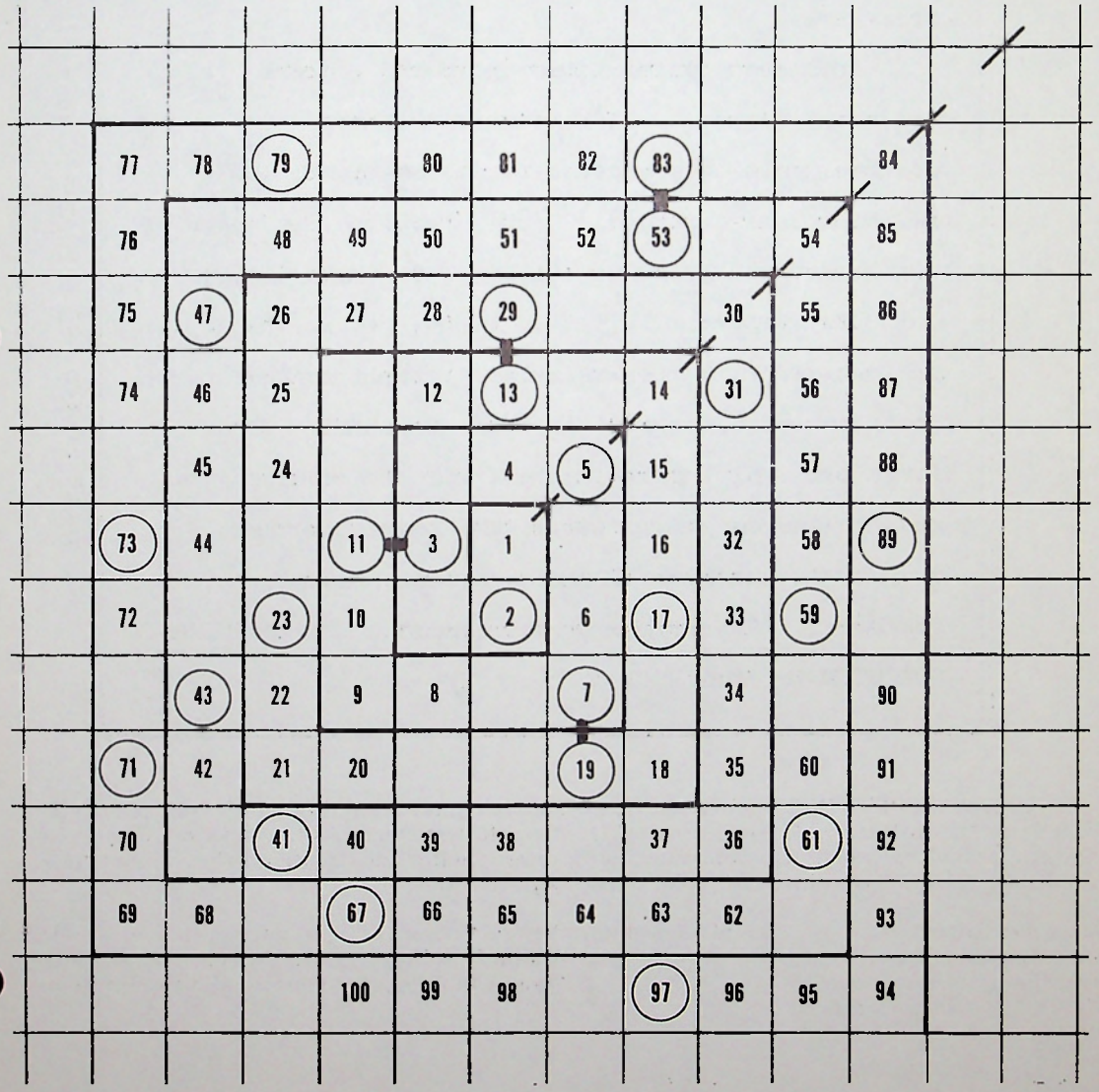
35

February 1976 Volume 4 Number 2

Popular Computing



Contest 4

In the diagram on the cover, the consecutive integers are entered into a spiral of squares, starting with one at the center, and proceeding clockwise.

For every prime number (circled) entered, a square is skipped. If a prime falls next to a previous prime (as happens four times up to 100), two squares are skipped. (The "next to" is taken only horizontally or vertically, not diagonally.)

The sequence 1, 5, 14, 30, 54, 84,... (that is, the numbers in the squares that proceed northeast from the center square) is to be extended. The first (and only) prize in this, our 4th contest, goes to the one who produces the longest correct list of the contents of those squares, done by computer. All entries must be received by POPULAR COMPUTING by April 30, 1976. □

The prize: $25, or a two-year subscription.

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $18 per year, or $15 if remittance accompanies the order. For Canada and Mexico, add $4 per year to the above rates. For all other countries, add $6 per year to the above rates. Back issues $2 each.

The CSR Function

(Continued Square Root)

The function defined as follows:

$$A = \sqrt{a_1 + \sqrt{a_2 + \sqrt{a_3 + \sqrt{a_4 + \sqrt{a_5 + \sqrt{a_6 + \sqrt{a_7 + \cdots}}}}}}}$$

(see PC34-10) provides some interesting computations, using nothing more advanced than square root and addition. The accompanying table shows the value for A for various sequences of a's, calculated by Herman P. Robinson on his Wang 720C.

Consecutive integers	1.75793 27566 18004 53270 88196 38218 13852 76531 99922
Even integers	2.15847 68723 11039 76565 58534 79807 02524 16696 94440
Odd integers	1.85025 31288 25914 28891 21451 97359 99374 79416 17995
Fibonacci	1.66198 24623 27811 55796 76060 81815 13129 50561 67562
Powers of 2	1.78316 58092 64098 88271 03049 92255 00328 85836 79511
Squares	1.94265 54227 63987 32822 14132 91412 66723 76880 73630
Cubes	2.17678 85971 33441 98583 98570 61721 83836 53483 24865
4th powers	2.46740 45317 17793 10674 79906 49563 77992 98547 96392
5th powers	2.82348 15128 34203 37757 13444 04555 84089 68438 53644
10th powers	6.05181 33295 21526 60210 47308 44812 93383 74146 88353
all ones	1.61803 39887 49894 84820 45868 34365 63811 77203 09180
1,4,2,8,5,7 repeating	1.87349 51093 71315 48791 93475 30993 64755 34321 31036
Primes	2.10359 74963 39897 26261 99396 49685 32544 40421 62288

Aaron Herschfeld, in the American Mathematical Monthly, Vol. 42, 1935, pp. 419-429, showed that a sequence will converge if

$$\lim_{n \to \infty} (\ln \ln a_n - n \cdot \ln 2) < +\infty$$

Thus, a sequence with terms increasing no faster than x^{2^n} will converge.
If the terms are exactly x^{2^n}, the CSR is $x\tau$, where τ is the golden mean. The sequence is started with n = 1.

Mr. Robinson points out the paradoxical situation that as the terms grow faster in size, fewer terms are needed for convergence, in general. An example illustrates this:

Let a given term be 10^{100}. The following term will be $k \cdot 10^{100}$, but the square root will be $\sqrt{k} \cdot 10^{50}$. Dropping this term introduces an error of the order of $\sqrt{k}$ parts in 10^{50}, if k is not too large. Successive square roots further reduce the error drastically. In the case of the factorials, starting with n = 17 gives an answer with a fractional error of the order of $2 \cdot 10^{-51}$. Robinson estimates that starting the factorial calculation with 35! will give a result good to more than 290 decimal places. □

N-SERIES 35

Log 35	1.544068044350275635498477363868143166715382514861857
ln 35	3.555348061489413679706112076669367369162686083850379
$\sqrt{35}$	5.916079783099616042567328291561617048415501230794340
$\sqrt[3]{35}$	3.271066310188589728224806902392531344098903147778906
$\sqrt[5]{35}$	2.036168004640398017360874164145317694261816167578535
$\sqrt[10]{35}$	1.426943588457650983590049861650304288871125929977205
$\sqrt[100]{35}$	1.036193062888396153570156125080192301740086857188821
e^{35}	1586013452313430.728129644625774660125176203950134526 1542666970224528012046269232516410 62
π^{35}	251330702007364298.616088947097500656998329124588764600253672035657158609090814251393 85
$\tan^{-1}$ 35	1.542232668956136624759150722706513354393176599772952

Bracketing

Many computing problems can be solved by means of a method called bracketing. The method (like bubble sorting) is inelegant, crude, unsophisticated, and inefficient. On the other hand, it is delightfully simple, easy to code in any language, and applies to a broad range of problem situations. Moreover, it is widely used, and knowledge of it should be made available.

It is best explained with an example. Suppose we are using a computer with a 32-bit word size. The high order bit of a data word is used to indicate the sign, leaving 31 bits for the size of the number. Thus, on such a machine, the largest positive integer that can be contained in one word is 2,147,483,647--one less than the 31st power of 2.

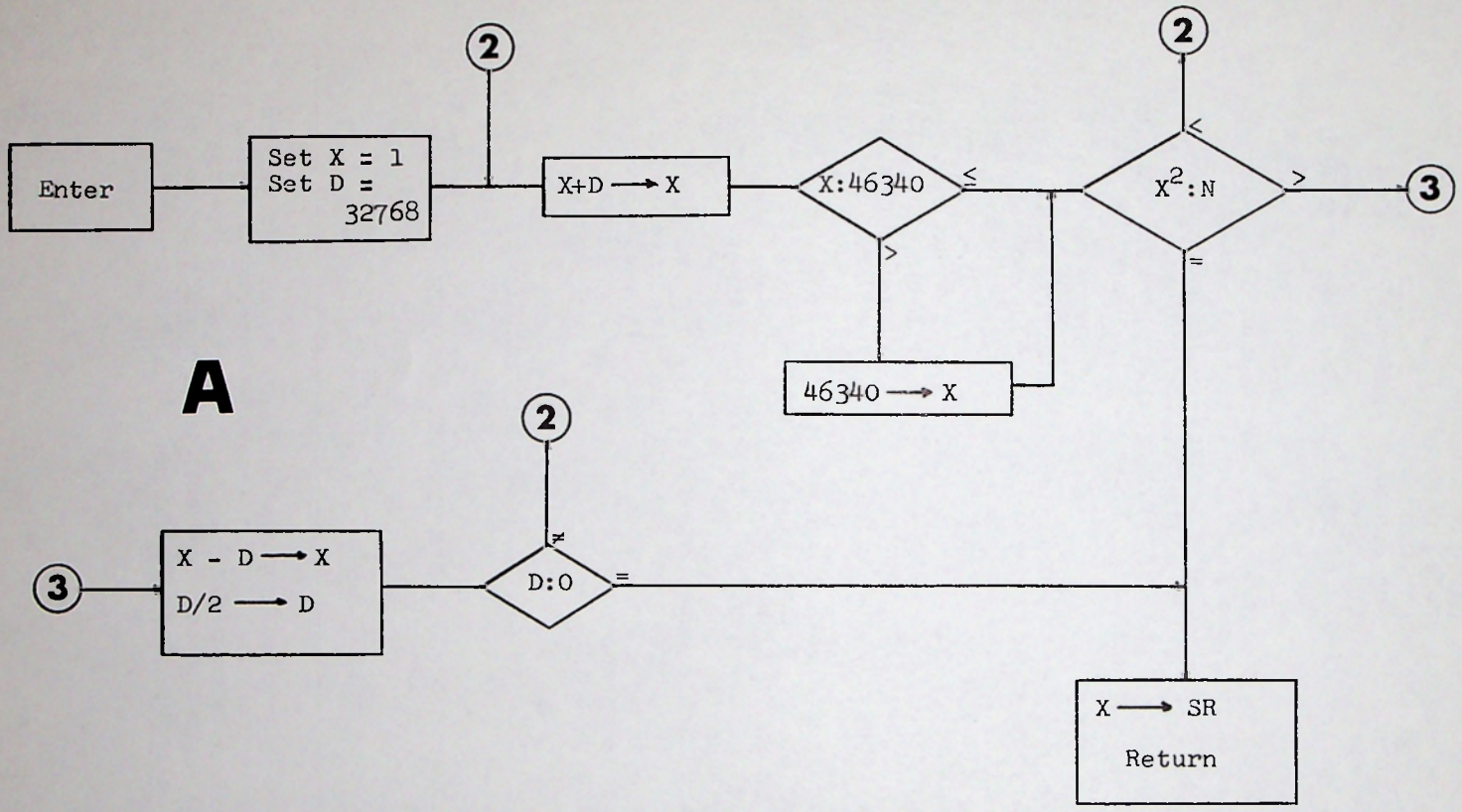
We wish to construct a subroutine that will accept a number, N, and produce in a word addressed at SR the square root of N. The range on SR is zero to 46340. A perfectly logical algorithm, then, would be to try every value between 0 and 46340 to find which one, when squared, came closest to N. If N is around two billion, the algorithm will require 46000 traverses of its loop in the worst case, which is hopelessly inefficient.

We should take bigger steps to start with, and then cut down the step size as we approach the root. The scheme shown in flowchart A will do nicely, keeping in mind that we are dealing entirely in integers. The value picked for the initial step size, 32768, is the largest possible power of 2 within the permissable range of SR. Since we choose to cut the step size in half each time we pass Reference 3*, we will pass through Reference 3 not more than 15 times to locate any root to the nearest integer.

This is the simplest application of the bracketing process (and there are, of course, much better algorithms for square rooting). We can see the essential elements of the process, however, from the simple example:

---

*The method resembles interval-halving here, but the two should not be confused. In general, interval-halving procedures are more difficult to program. A discussion of interval-halving can be found in Introduction to Computer Science, by Harry Katzan, Jr., Petrocelli/Charter books, 1975.



A subroutine to calculate the square root of the integer addressed at N by bracketing. The result is addressed at SR. The horizontal arrow means "replaces."

1. The solution space should be known; that is, the total range within which the result must lie.

2. We arrange to explore that space, starting with very large steps.

3. The key element is the decision (Reference 4 in flowchart A). We must have a way to determine when we have gone passed the desired result. Notice that this matter is not self-correcting. If the result is passed and that fact is not correctly noted, the process will hang up and/or yield erroneous results. When the decision says that we have gone too far, we back off one full step.

4. We arrange to cut down the step size by any convenient amount. (Most frequently the step size is reduced by a factor of 10.)

5. We arrange a graceful way to terminate the process when we have reached the desired level of precision.

Like all numeric processes, bracketing can not be applied blindly. The details will differ from problem to problem, but the overall plan is as stated.

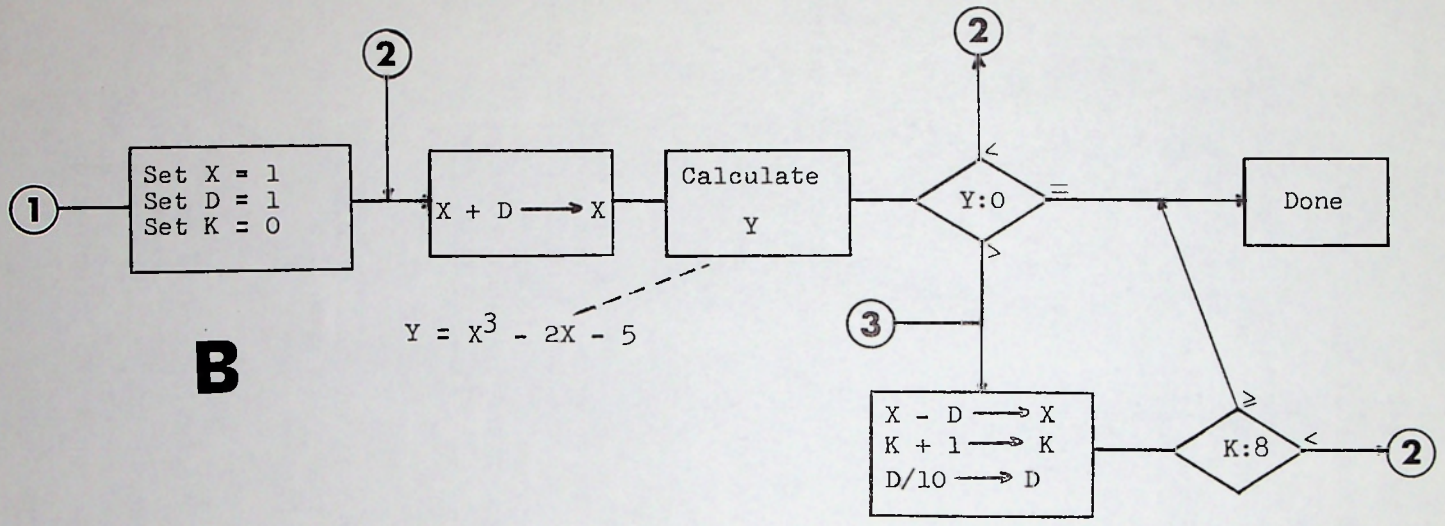
We now have a new tool, and we have tried it out on a known case.

Suppose now that we are working in scientific notation, so that the range on N is from zero to $9 \cdot 10^{99}$. We could apply the same procedure, but our initial step size would have to be something like 10^{99}, and in the worst case we would go around the loop 332 times. The bracketing process does not apply in this case.

We can, however, apply it to other problems quite profitably. To find the real root of Wallis' equation ($X^3 - 2X - 5 = 0$) for example, we have

$$Y = X^3 - 2X - 5$$

and the flowchart logic B applies. Preliminary analysis tells us that the real root lies between 2 and 3, which is our solution space. A counter, K, will terminate the process after 8 traverses of Reference 3, for a solution correct to 8 significant digits. The termination procedure could be any test for convergence, but it is simpler to count through the loop.



The bracketing process applied to calculating, to 8 significant digits, the real root of

$$Y = X^3 - 2X - 5.$$

The process lends itself to problems in maxima and minima. For example, consider the old problem of constructing, from a sheet of tin of length L and width W, an open topped box of maximum volume. The logic of flowchart C shows the bracketing process applied to this familiar calculus problem. The volume calculated for each trial value of X is compared to the previous volume, ØV. As long as the volume keeps increasing, we are heading in the right direction. As before, when the volume drops ("too far"), we back off one full step. But now there is a new problem. The various cases that exist in seeking a maximum or minimum can cause trouble. For safety, the box at Reference 3 should be made X - 2D ⟶ X, to back off <u>two</u> full steps.

Figures P, Q, and R show what can happen in seeking a maximum (indicated by the arrows). P is the normal case, in which a drop in the calculated value indicates correctly that the maximum has been passed; in this case, a retreat of one step would suffice. But in the situations shown at Q and R, although the drop does indicate that the peak has been passed, a retreat of one step would not bring us past the peak, and the process would be out of control.

Again, we have applied the new tool to a problem that is better handled with calculus. But the new tool now allows us to operate easily in cases where a solution by calculus might be awkward. Consider, for example, the problem given in issue No. 16, page 15: To divide the number 10 into two parts so that when the first part plus its square root is multiplied by the second part plus its square root, the product is 23; that is, to solve:

$$(X + \sqrt{X}\,)(10 - X + \sqrt{10 - X}\,) = 23.$$

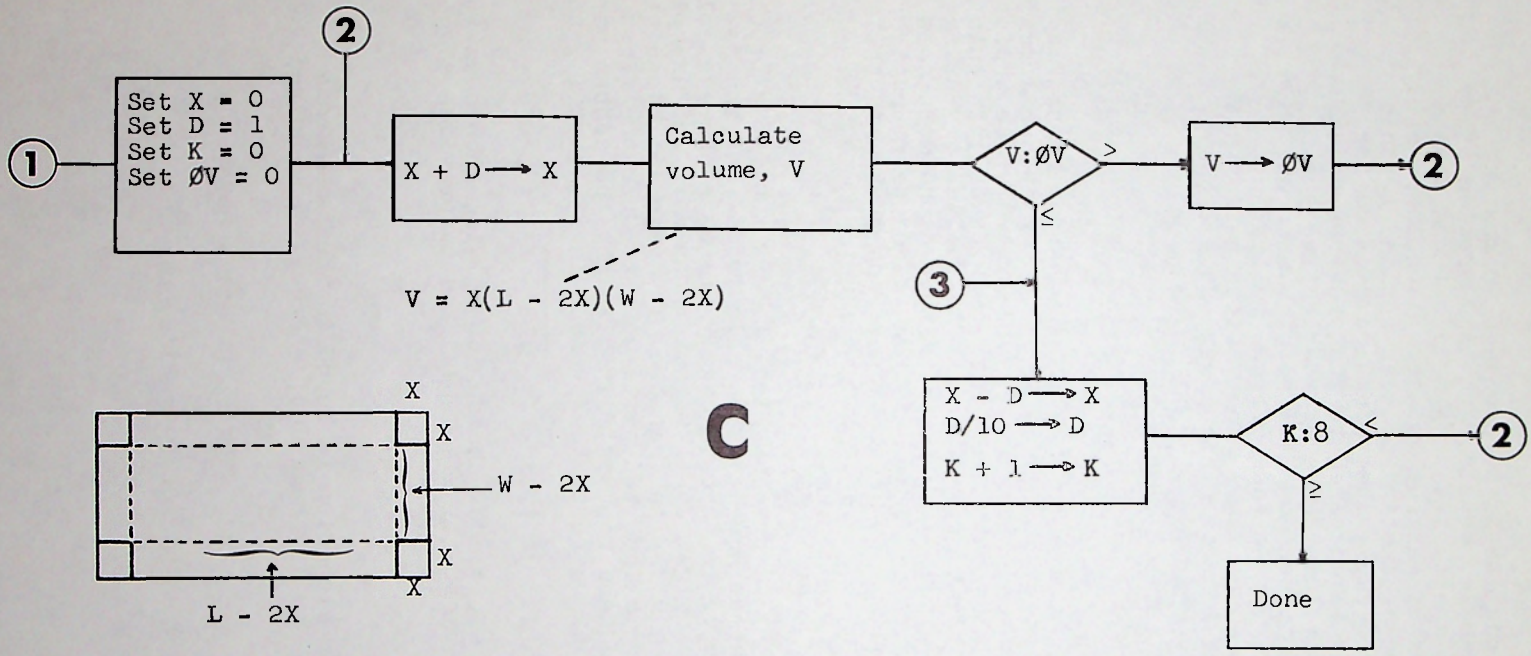
There are efficient schemes for solving such equations, but bracketing will also work, and it provides a quick and dirty solution.

A similar situation is found in the Lake/Fence problem (issue No. 15, page 10), where it is required to find the maximum of

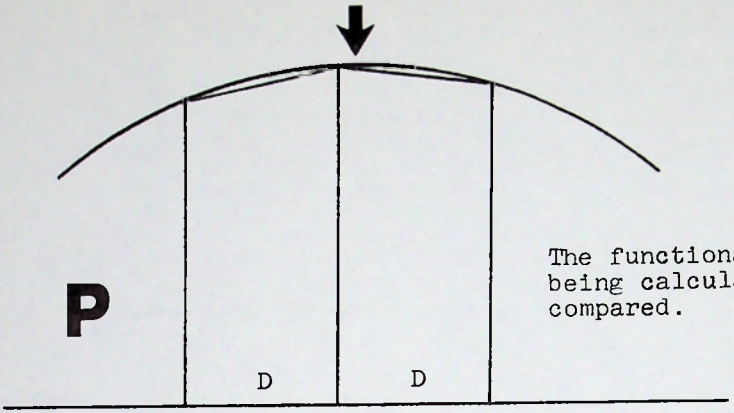
$$A = X(500 - 2X) - 5280^2T + H(250 - X)$$

where $T = \tan^{-1}\{(250 - X)/H\}$

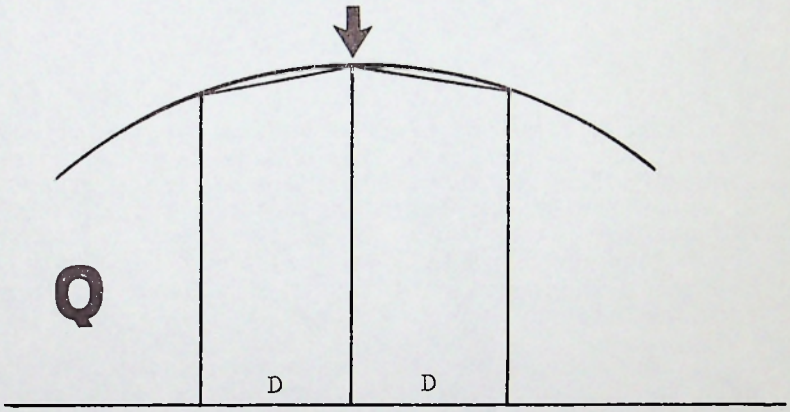
and $H^2 = 27815900 + 500X - X^2$

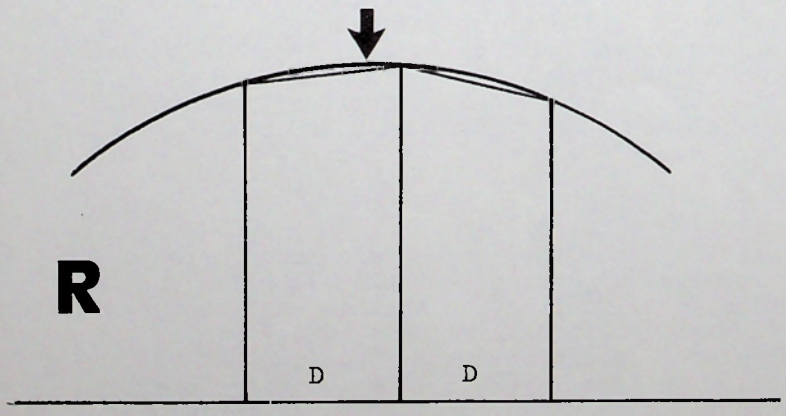


The bracketing process applied to the problem of the open-topped tin box, a simple problem in maxima.

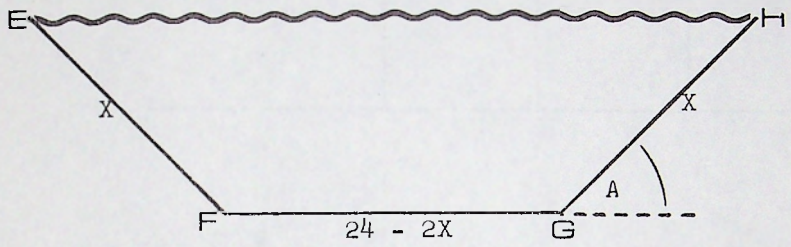


The functional values being calculated and compared.





The bracketing process can even be applied (although it is tricky) to more complicated situations involving several variables. The problem of Snoopy's water dish, for example (see Figure S) calls for the dimensions of a dish in which the length of the cross section (E-F-G-H in the Figure) to be 24 inches, and the length X and the angle A are to be determined so that the dish will have maximum volume.

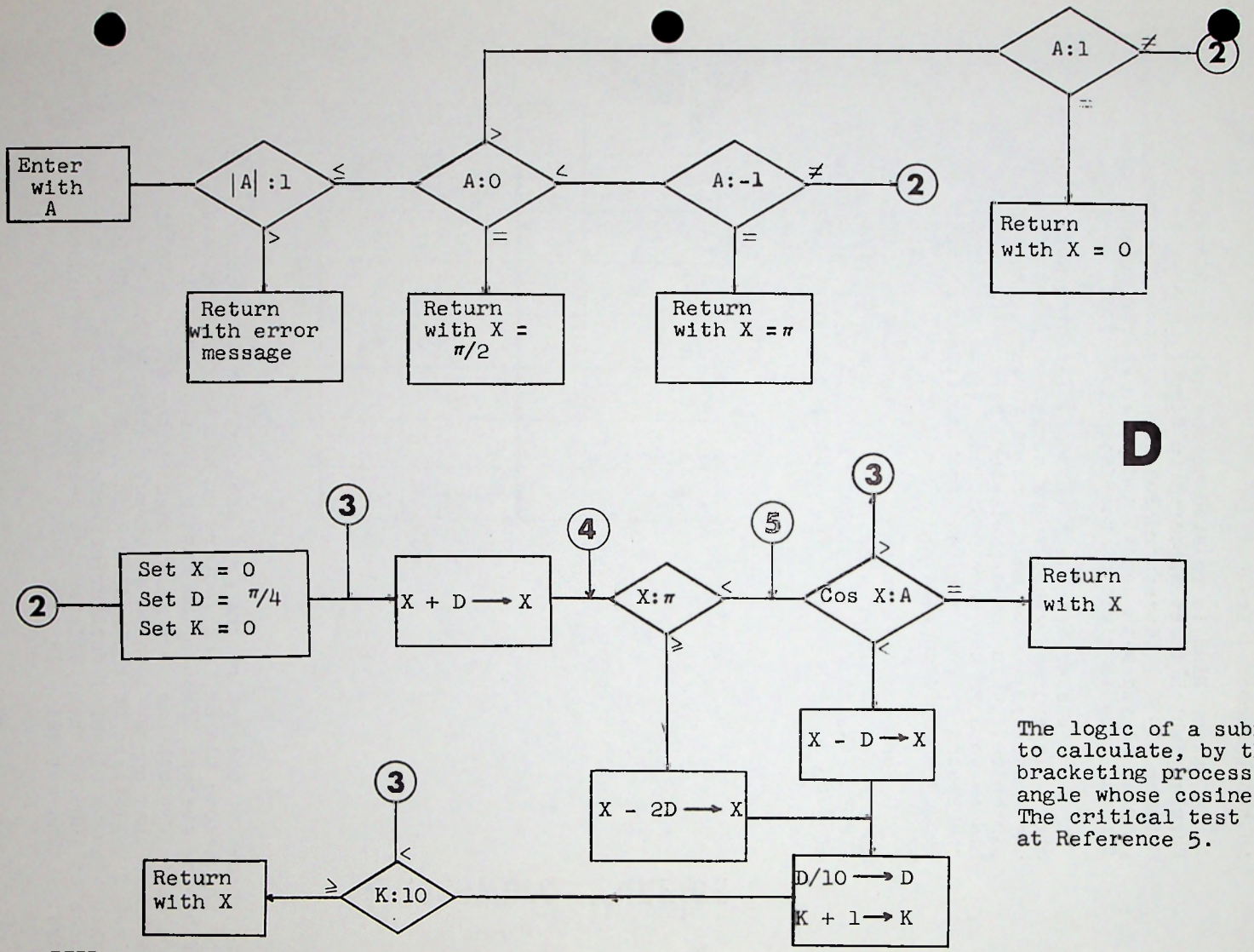


S

The bracketing process is most useful for inverting a function. To take a simple case, suppose we need the arccosine function and we are working in Fortran, which gives us the direct function, cosine, but not the inverse. Flowchart D shows the logic of a subroutine for which the input is a number A between $\pm$ 1, and the output, X, is the angle whose cosine is A. (Flowchart D is modified from one made by Robert White).

The subroutine logic begins by editing the input data for validity and then immediately takes care of the obvious cases. The bracketing process begins at Reference 2, and the critical element is the comparison of cosine X with A at Reference 5. The test at Reference 4 is inserted as a safety factor, to insure that the process does not exceed the range of the principal value for arccosine. The scheme shown could be further improved by capitalizing on the symmetry of the cosine function in the first two quadrants. It might be further improved by testing for other common values, such as 0.7071067 or 0.5 for A, giving "exact" values of $\pi/4$ and $\pi/3$ respectively.

The function arccosine can be calculated more directly by formula, of course, but the algorithm applies to any function. For a function that is analytic, there are usually analytic methods available to invert it, but the bracketing process can invert a function that is discontinuous. One can invert a life insurance premium table, or a postal rate scale, or almost any function that is defined in one direction. □



The logic of a subroutine to calculate, by the bracketing process, the angle whose cosine is A. The critical test is at Reference 5.

PC35-13

The game of Master Mind has been featured in Games & Puzzles since 1973, and was written up in Time in the December 1 issue.

Master Mind is related to the game of Guessword Puzzles, which first appeared in Computing News in October, 1954. Guessword Puzzles is a game for two people, played with pencil and paper (a convenient grid for the game is given on the facing page).

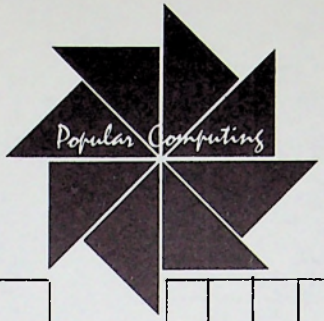
One player selects a (hidden) 7-letter word. The other player then guesses any number of 7-letter words, one at a time. For each word he guesses, the first player tells him how many letters agree with those of the hidden word in the same position. From the numbers that the second player receives from the first, he must deduce the hidden word.

For example, suppose the hidden word is SUCCUMB. If the second player guesses successively GENERAL, ELEMENT, ADAMANT, and BANANAS, he will be given counts of zero in each case. The situation is now:

Guessword Puzzles

GE AB	EL D	NE A	EM A	RE AN	AN	LT S	
G	E	N	E	R	A	L	0
E	L	E	M	E	N	T	0
A	D	A	M	A	N	T	0
B	A	N	A	N	A	S	0

The spaces at the top of the grid provide room to record, for the second player, the letters he has so far eliminated. If he now guesses UNUSUAL, he gets a count of one. Since the AL in UNUSUAL has been eliminated by the AL of GENERAL, he knows that he has located a correct letter within the first five letters of the hidden word. A guess of UNTRIED (with a count of zero) further narrows the choice, and a guess of TRUSTED then isolates the second U of the hidden word. The situation is now:



PC35-15

5

10

15

20

25

GE AB U	EL DA N	NE AT	EM AR	RE AN I	AN E	LT SD	
G	E	N	E	R	A	L	0
E	L	E	M	E	N	T	0
A	D	A	M	A	N	T	0
B	A	N	A	N	A	S	0
~~U~~	~~A~~	~~N~~	~~A~~	(U)	~~A~~	~~L~~	1
U	N	T	R	I	E	D	0
T	R	U	S	T	E	D	0

At this point, most of the available information has been used up, and the game continues. If the next guess was ICICLES (for a count of one) a good player would probably deduce SUCCUMB in four more guesses, for a score of 12 for the game.

Since the game is active for one player and passive for the other, it becomes expedient to play two games simultaneously, and the playing grid is constructed for that purpose. Par for the game is about 20 guesses, and a game usually takes about 45 minutes.

The two lists below are guesses for two different games, which refer to two different hidden words. The count for every word in the list on the left is one; the count for every word in the list on the right is zero. What are the two hidden words?

ICICLES	MINIMUM
MINIMUM	ICICLES
SUPPORT	GENERAL
GENERAL	ELEMENT
UNUSUAL	UNUSUAL
ADAMANT	FUTURES
UNIFORM	ODOROUS
DIAPERS	ADAMANT
CRANIUM	BOLOGNA
PRESORT	BANANAS
BANANAS	QUICKLY
AVERAGE	OBLOQUY
FUTURES	DIAPERS
ABSOLVE	ROADMAP

□

PROBLEM 117 Solution

Problem: A subroutine in a program produces as its output 3-digit numbers (N) in no special order and with possible strings of duplicates. A second subroutine is needed which will examine these numbers as they are produced and report each new longer string of duplicates.

Counting String Lengths

The flowchart shown below gives the required logic. A counter, C, keeps track of the string lengths. The counter is initialized to zero in the housekeeping phase of the main routine, and the values of old N (ØN) and old C (ØC) are also initialized to zero. The second subroutine is called each time a new value of N is generated by the first subroutine.

To test this logic, generate the sequence

1,2,2,3,3,3,4,4,4,4,5,5,5,5,5,4,4,4,6,6,6,6,6,6,...

and verify the following expected output:

1	1
2	2
3	3
4	4
5	5
6	6

and so on □

ANNUAL INDEX

Volume 3, 1975 Issues 22 through 33 206 pages